CHALMERS

Automatically generated user interface for an XML editor

Automatically generated graphical user interface for use with XML configuration files

Master of Science Thesis in the Programmes *Networks and Distributed Systems* and *Secure and Dependable Computer Systems*

ERIK ANDERSSON

JOACHIM LUNDGREN

Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2010

Automatically generated user interface for an XML editor
Automatically generated graphical user interface for use with XML configuration files

ERIK ANDERSSON,
JOACHIM LUNDGREN

Examiner: JAN JONSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering

SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2010

Abstract

Developing software systems are costly, with this in mind you would want to deliver one system to several customers instead of developing a new system for each customer. The customers' needs and demands are often very different, so a system needs to be flexible to fit many potential customers. There are also situations where a customer's need might vary over time. Many hours can be saved by designing your system so that it can be reconfigured to fit your customers. In this paper we will present how we created tools to simplify editing of XML configuration files. We put in great effort to keep our components as modular as possible, so that they easily could be added to different projects without modification. We will also present a few examples on how our components could be used.

This master thesis was conducted at Saab Security's division for Air Traffic Management, ATM, in Gothenburg and the final components are specialized to fit their needs. The components are general enough to work with any well formed XML document, but some features will only be available when the documents follow certain patterns and schemas used by Saab Security.

During this master thesis we have developed using the development method called Scrum. In this report we will also give an explanation of Scrum and how we have benefited from working with a well known development method.

Table of contents

List of abbreviations

Abbreviation	Description
API	Application Programming Interface
ATC	Air Traffic Control
ATM	Air Traffic Management
COM	Component Object Model. Language independent technique for communication between objects or program.
CSS	Cascaded Style Sheet. Used to assign design parameters to structures.
CSV	Comma-Separated Values. File format for storing data in a table.
GUI	Graphical User Interface
MFC	Microsoft Foundation Classes. A library for creating GUIs in Windows.
STD	Software Test Description. A document containing information about tests and how to perform them.
STR	Software Test Report. A document used when performing the tests in one or several STDs.
WPF	Windows Presentation Foundation. Graphical subsystem for rendering user interfaces.
XAML	Extensible Application Markup Language. XML-based language used to structure values and objects.
XML	Extensible Markup Language. A set of rules for encoding documents electronically.
XSL	Extensible Stylesheet Language. Languages used for transforming and rendering XML documents.

1 Introduction

Saab Security is a division of Saab AB working in the business area of civil security. This thesis was conducted at the sub division that develops air traffic management (ATM) systems. Saab Security delivers ATM solutions to several airports, both civilian and military. Many of Saab Security's ATM systems utilize the XML format in different ways.

The XML format has been around since the late 1990s [1]. XML is used to organize data into an easily parsed structure. The XML format is extremely flexible, allowing the author of a document to define its own format. There are a lot of tools to create, manipulate and access the data in an XML document available.

Today the XML format is used by a large number of programs and projects [2]. There are also a lot of standards based on the XML format, e.g. RSS [3], XHTML [4] and OOXML [5].

This master thesis has been conducted in-office at Saab Security and aims at simplifying the handling of the XML files used there. The project has two main goals: to create components for XML editing applications and to evaluate the Scrum development method when used in a semi academic project.

1.1 Background

ATM systems have to be flexible and configurable to fit the customers' different needs. The large amount of parameters makes the systems difficult to configure. Today, at Saab Security, this configuration is done in several XML-files. Recently an effort to generate schemas to go along with these XML-files has been performed. To generate the schemas a specialized XML file, called rawItXml, is used to describe the contents of the schema, this file is then used to create the actual schema file. We took advantage of both the rawItXml and the actual schema file when creating our components.

Editing these configuration files, for each new customer, is time consuming and costly. Since the configuration files are extensive and detailed, a developer with system knowledge is often required for the job.

1.2 Purpose

This thesis project has two main goals, presented in detail below. We have chosen to clearly separate the actual implementation, i.e. coding, from the more academic research part. We believe that it will be easier to get a good overview of the project this way. Another benefit is that a reader that is only interested in one of these parts will have an easier task of sorting out the interesting material and results.

1.2.1 Goal 1, creating a set of software components that simplify XML editing

The product requested by Saab Security is a set of reusable software components that simplifies the editing of XML files. The controls should present the XML data to the

user in a simple and intuitive way and should not require the user to have any understanding of the XML format.

The components should be reusable in different contexts, both in graphical representation and non-graphical representation. The components should be written in a way that allows for editing of any well formed XML file, although specialized functionality might require a schema or layout used by Saab Security.

The components should be well documented and easy for other developers to understand and use in their own applications.

The result of this goal will be a working software component library that simplifies system configuration for Saab Security.

1.2.2 Goal 2, evaluating Scrum in a semi academic project

To organize the work during this project we have chosen to use a software development method called Scrum. Scrum is an agile development method suitable for smaller development teams.

A thesis is more than just software development, even if the goal is to produce a software product. Because of this we will adapt the Scrum method to incorporate the more academic parts of the project, such as research, report writing and administrative parts.

Throughout the project we will continuously evaluate the method. This report will present both the continuous evaluation results and the conclusions at the end of the project.

The result of this goal will answer the question: “How well does Scrum work in a project with both academic and industrial parts?”

1.3 Organization

Chapter 2 of this document will present the methods used to perform this thesis. It includes details about the choices of programming language, development environment and development method.

Chapter 3 presents the theory needed to understand the project and the report. It will include descriptions of the key concepts of the project and information specific to Saab Security. It also includes studies performed as well as description of the products developed.

Chapter 4 describes the results of the project including detailed description of the components created.

In Chapter 5 we discuss the results produced during the project, in relation to goals, theory and methodology.

In Chapter 6 we present the conclusion we have drawn from the project results.

The report ends with a list of references and an appendix section containing unprocessed information that can help the reader get a deeper knowledge of certain parts of the project.

1.3.1 Sensitive information

Since Saab Security deals with confidential material from both military and civilian parties we have avoided presentation of sensitive data and sometimes obfuscated information to remove any potential information leakage. This has been done in a way that does not affect the results or conclusions of this report.

2 Method

During this master thesis we used Microsoft Visual Studio 2008 as our development environment and wrote our code in C#.NET.

The project was conducted using Scrum [6] as development method. The reason for choosing Scrum was that our supervisor at Saab Security suggested it early on. Since we had already considered employing a development method this felt like a good choice.

Using Scrum means that we planned the work in three week segments called sprints. During each sprint we focused on a number of tasks selected from a backlog. We decided to incorporate writing of this report into each sprint to ensure progress throughout the project.

2.1 Scrum

As we mentioned earlier, we used the development method called Scrum during this master thesis. Using a known development method gave us continuity in our work and forced us to stay on track and always work with high priority tasks. Using Scrum we divided our work into small manageable parts, which speeded up our work a great deal.

2.1.1 Overview of Scrum

Scrum is an agile software development method from the early 1990s. The development is done in short iterations called sprints. Each sprint consists of small tasks with an estimated time to complete. By keeping tasks small it's easier to understand how to complete the task and the work speed thereby increases. Having small tasks also makes it easier to see that the project is constantly moving forward.

Scrum has four important artifacts. The *product backlog* is a list containing everything that might be needed in the product. Everyone can add features, functions, bug fixes etc. to this list, but only the product owner can change the priority of the attributes in the list. For each sprint, a set of tasks are selected from the product backlog and placed in the *sprint backlog*. The sprint backlog contains tasks to be performed during the current sprint. To measure how much work is remaining of a sprint, a *sprint burndown* chart can be used. A *release burndown* chart is the same as the sprint burndown, but for the whole project.

2.1.2 Scrum roles

In Scrum there are a few defined roles. The most important roles are the *product owner*, *Scrum master* and the *Scrum team*. There are also other roles that need to be considered, although they are not a part of the actual Scrum process, namely *stakeholders* and *managers*.

2.1.2.1 Product owner

The product owner is the customer or someone who represents the customer. His role is to decide what to prioritize from the product backlog. By doing this he ensures that the developers focus on the most important features. The product owner is also responsible

for reviewing the system at the end of each sprint. During a sprint, the owner can't interfere with the team's work.

It is important that the product owner is one person, not a committee. However, a product owner might be influenced or receive advice from other stakeholders.

2.1.2.2 Scrum Master

The Scrum master ensures that the team follows Scrum rules and guidelines. A Scrum master should be seen as a coach, not a boss, to the Scrum team. It is not his job to tell the team what to do, the team should be self-organized. In small teams, the Scrum master could be a part of the Scrum team.

2.1.2.3 Scrum Team

The Scrum team is a self-organized and cross-functional development team. Cross-functional means that each member of the team should have all the necessary skills needed to create an increment of work.

The team often consists of 5-9 people and their job is to realize the sprint backlog. How this is done is up to the team itself, no one control them, not even the Scrum master.

2.1.3 Scrum process

The Scrum process consists of short iterations called sprints. During each sprint, the developers work with the tasks that have the highest priority. At the end of each sprint the project is supposed to deliver a solution that could possibly be shipped to a customer. This results in that the customer can take advantage of the software early in the development period.

2.1.3.1 Sprint planning meeting

The sprint planning meeting is held in the beginning, as a start, of each sprint. At the planning meeting, the Scrum team and product owner determines the goals for the upcoming sprint. The planning meeting is often divided into two parts, one "what" part and one "how" part.

During the first part the product owner presents what tasks in the backlog that have highest priority. Together with the team they select what tasks that should be realized during the upcoming sprint. During the second part, the team sits down and decides how to realize these tasks.

2.1.3.2 Daily Scrum meeting

Each day the team meets for a short meeting. The purpose of this meeting is to synchronize work and to give a brief progress report. Each member of the team should, one at a time, explain what they have done since last daily meeting, what they will do between now and the next daily meeting and what obstacles prevents him or her from accomplish his or her goals.

2.1.3.3 Sprint review meeting

The sprint review meeting is a meeting to inform everyone involved how the project is moving along and what have been done during the sprint. This meeting is important for everyone that should have influence on what the final product should look like. The sprint review meeting gives valuable input to the upcoming sprint planning meeting.

2.1.3.4 Sprint retrospective meeting

After the sprint review meeting, but before the next sprint planning meeting, the team has a sprint retrospective meeting. The purpose of this meeting is to reflect on the sprint and see what process improvement that can be made.

2.1.4 Our implementation of Scrum

Since all development projects are different, it is important to adjust the process to the situation. We implemented Scrum with sprints of three weeks. Due to the fact that our team only consists of two members, we had to do some modification but the key concepts were still the same.

There are a lot of fixed numbers in Scrum, such as team size and meeting length. However since our team is very small we decided not to go the full length of all meetings. This gave us more time to do the actual development in each sprint.

The fact that each sprint should result in a shippable version of the product gave our employers a guarantee that, even if we failed to complete the thesis, they would receive a product they could benefit from. Since we included documentation in each sprint, the product would have been well documented.

3 Theory

This section presents the theory needed to understand the different parts of the project. It will also present background studies and research conducted during the project.

3.1 Concepts specific to Saab Security

There are a few concepts and products that are specific to Saab Security. Those of importance to this project will be described below.

3.1.1 rawItXml

rawItXml is an XML schema format that has been developed by Saab Security. The format is used to generate custom schemas suitable for the XML configuration structures used by Saab Security. The rawItXml-file is generated by a certain software component to describe how data is to be organized. An XML schema is then generated from this information.

In our project we will add support for utilizing both rawItXml files and XML schema files.

3.1.2 ComCore

ComCore is a communication platform used by some of Saab Security's systems. The platform is based on a self configuring network protocol that both monitors the connected computers and ensures reliable information delivery. The system adds redundancy functionality that guarantees full system functionality even after the loss of one of three servers.

3.2 Viewing a Windows Form GUI-controller in a MFC-application

One important condition for the components is the ability to integrate with both .NET applications and C++ applications that use MFC [7].

3.2.1 Why is it important to run our modules outside .NET

Today many of Saab Security's applications are written in C++ using MFC. MFC is a collection of object-oriented class libraries, which were developed by Microsoft for programming graphical applications. To be able to integrate our components with existing, or future, applications it was preferable that our code could be run in C++. We tried two possible approaches when trying to integrate .NET code into an MFC application.

3.2.2 How to achieve compatibility with MFC

The first approach we tried was to create COM components that could be drag and dropped into an MFC application. This would allow other applications to take full advantage of our components. To do this, one must compile with the command com interop [8] set to true. When digging a bit deeper it got clear that this modification was not the only thing needed. To create a COM object several utilities where needed, such

as a utility to generate a globally unique identifier (guid.exe), creating a strong name (sn.exe) and an assembly registration tool (regasm.exe). It suddenly became very complex and another solution would be preferable.

The second approach we tried was to have a button or link on the MFC-application, which started the .NET form [9]. With this approach, it is possible to send arguments from C++ to the .NET form and to receive information from the .NET form to the C++ program just as if it was an ordinary MFC dialog. This means we can integrate our components into other applications.

The only drawback with this method is that you can't add a component directly into an MFC view or dialog, but the need for that was considered negligible.

3.3 Background study on existing XML editors

This section will present the result of the background study on existing XML editors available on the market.

The purpose of this background study is to investigate how conventional XML editors present the data and what tools they provide for editing. This information will be used as input to our own application and components. We will focus on collecting only the information that is relevant to us from each evaluated editor and summarize this in a way that gives a clear view of what parts we can use or cannot use. In addition to this we provide some first impressions of the editor being evaluated.

3.3.1 Liquid XML Studio 2010

Liquid XML Studio [10] is a large program suite developed by Liquid Technologies Ltd. It is a proprietary software and ranges from approximately €80 to €600 depending on the edition.

The version used in this evaluation was *Liquid XML Studio Developer Edition (Trial) 8.0.7.1998*. A screenshot can be seen in Figure 4.1.

3.3.1.1 First impressions

When first starting the application the user is presented with a Visual Studio-like GUI. Liquid XML Studio uses a project based organization scheme allowing the user to bundle all relevant files in one place.

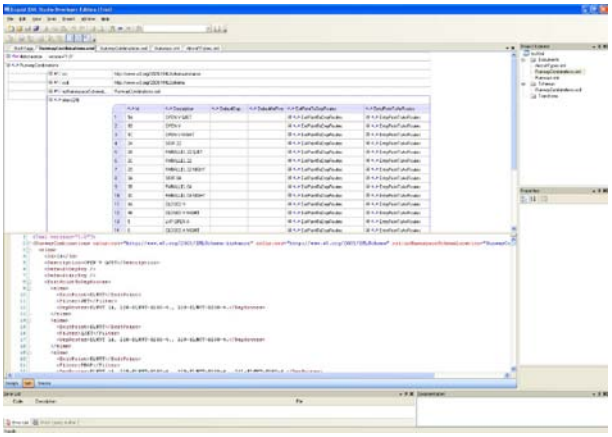

Figure 3.1. *Screenshot of Liquid XML Studio 2010*

After testing the application for some time it is obvious that it is not really tailored for editing XML files, but rather focused on creating XML files and generating XML schema files and even source code able to represent the document.

However, it is of course possible to edit files and tools are available to make editing much easier compared to text editors. The “design view”, which is available when opening an XML file, provides a very nice looking tree view of the XML file. It presents list of elements in tables and clicking an attribute allows for editing directly in the design view. When selecting an element or an attribute it is possible to see the corresponding XML code in a split window.

3.3.1.2 Relevance to our project

The design view available in Liquid XML Studio is certainly interesting for our project. It provides a very nice overview of the file and can be redesigned to fit our needs.

The tools for editing schemas and generating source code are irrelevant to this project and will not be considered.

3.3.2 <oXygen> XML Editor

<oXygen> XML Editor [11] is a part of a series of products developed by SyncRO Soft Ltd. <oXygen> XML Editor is a proprietary software and single user licenses range from approximately $60 to $450 depending on the edition.

The version used in this evaluation is *<oXygen> XML Editor 11.1, build 2010020412*. A screenshot is shown in Figure 3.2.

3.3.2.1 First impressions

When first starting <oXygen> XML Editor the user is presented with a user interface somewhat like the one used in NetBeans [12]. The files are organized into projects. There are a lot of features regarding transformations. There are also features allowing generation of XML files from Microsoft Excel documents and databases.

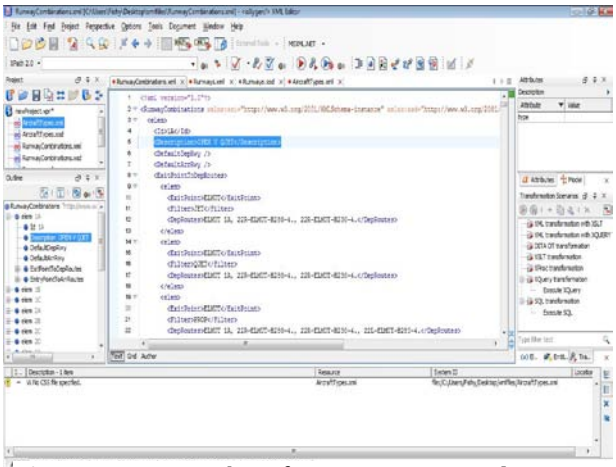

Figure 3.2. *Screenshot of <oXygen> XML Editor*

There are several views to present and edit the data. There is an outline section and a main section. In the outline section a tree representation of the XML file is presented. It is possible to filter and alter data directly from this view.

In the main view the user is presented with a choice of three presentation alternatives. There is a “Text” view that simply shows the raw XML text. There is a “Grid” view that presents the data in a more structured way, much like Liquid XML studio, using tables to represent lists etc. The third alternative is the “Author” view allowing the user to create a cascaded style sheet (CSS) to customize the presentation according to specific needs or preferences.

3.3.2.2 Relevance to our project

From the user interface there are not a lot of new inspiration since most of it resembles Liquid XML Studio, evaluated above. The only thing that stands out is the ability to filter what to show in the outline view, allowing the user to choose to show/hide text within elements and attributes of elements.

3.3.3 Altova XMLSpy

Altova XMLSpy [13] is an XML editor that has been on the market for several years. It is a proprietary application with a price ranging from approximately €130 to €800 depending on the edition.

The version used in this evaluation is *Altova XMLSpy 2010*. A screenshot is shown in Figure 3.3

3.3.3.1 First impressions

Altova XMLSpy is project based and supplies the user with a number of presentation sections. There is a main section showing the current file. The main section can be set to use a number of different modes. “Text”-mode shows the raw XML code, “Grid” view show the file in a tree like structure. In addition to these two there are 5 other view choices, but they require input from schema files or similar. One thing that stands out is that there are two sections that, when selecting an element in the main view, displays that node’s elements and attributes

respectively.

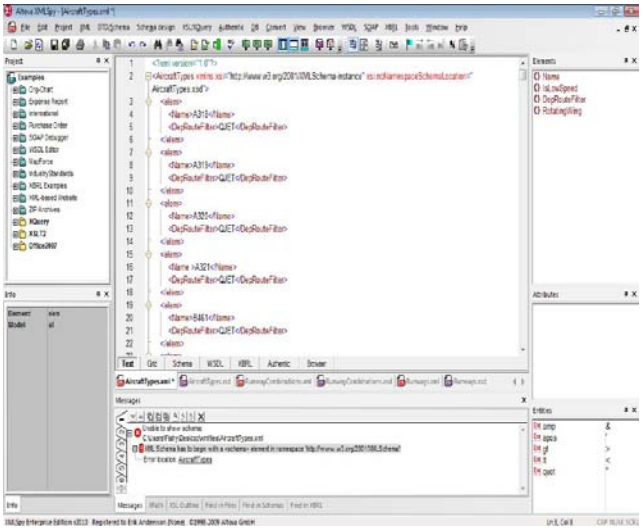

Figure 3.3. *Screenshot of Altova XMLSpy*

3.3.3.2 Relevance to our project

The two sections displaying attributes and elements could be interesting to look into for our project. They could be used as a stepping stone to accessing specialized editing tools for the specific attribute/element.

3.3.4 Syntext Serna Free

Syntext Serna Free [14] is as its name implies, free. Actually it is not only free, it is also open source.

The version used in this evaluation is *Syntext Serna Free 4.2.0-20091009.0*. A screenshot is shown in Figure 3.4.

3.3.4.1 First impressions

The main window in Syntext Serna has two display sections. On the left there is a “ContentMap” consisting of a tree representation of the XML file. The main view shows a parsed version of the file displayed as a document. This view is confusing and unintuitive at a first glance.

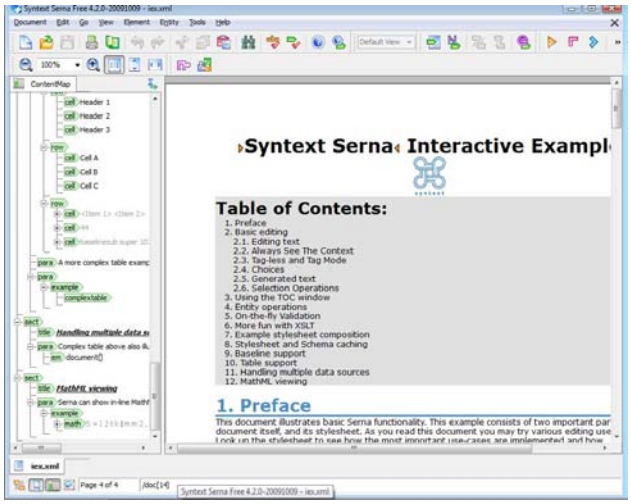

Figure 3.4. *Screenshot of Syntext Serna Free*

After some evaluation of the program it becomes apparent that it is designed to utilize templates to display the XML files in a customized and more comprehensible manner. There are built in templates for some of the common XML structures. If you don’t have a template matching your file the presentation in the main section will be very confusing and do more harm than good.

3.3.4.2 Relevance to our project

Since Syntext Serna is very dependent on templates to present the files in a user friendly way, it is not of much use as inspiration for our project.

3.3.5 XPontus XML Editor

XPontus XML Editor [15] is a free, open source, XML editor. It is a relatively young project with only two active developers.

The version used in this evaluation is *XPontus XML Editor 1.0.0.2*. A screenshot is shown in Figure 3.5

3.3.5.1 First impressions

XPontus is a quite simple editor. Basically it is a text editor that has been adapted to the XML format. It operates on one file at a time, not project based like its more advanced competitors.

The layout consists of two sections. On the left there is an “Outline” section which displays a tree representation of the XML file. On the right there is a main section with a color coded text representation of the XML file.

There are a couple of handy tools such as the ability to use XPath expressions and XQueries in the document. There are also tools for XSL transformations, schema validation and checks to see if the document is well formed.

3.3.5.2 Relevance to our project

XPontus has no ground breaking features or user interface. It is simple and effective but does not really stand out from the rest of the editors in functionality or visual presentation.

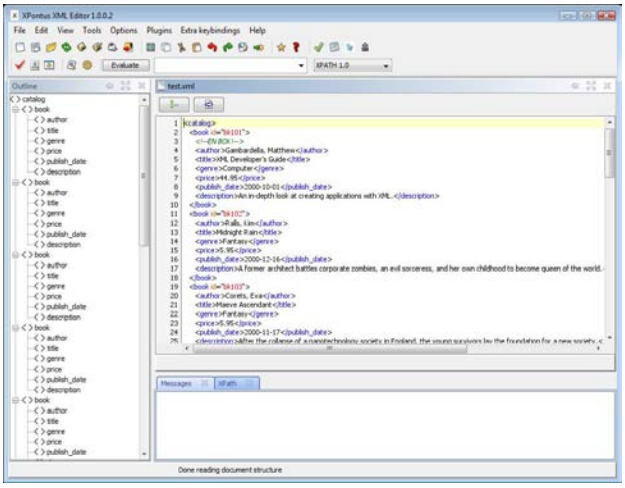

Figure 3.5. *Screenshot of XPontus XML Editor*

3.3.6 Summary

There are a lot of XML editors available for purchase and download. The complexity of them ranges from simple text editors with a few added features to large program suites with an enormous amount of tools and features.

During this evaluation it has become clear that existing XML editors offer little of the functionality we are striving for in our project. Most of the editors available require a quite extensive knowledge of the XML format and its support formats, e.g. XLST, XQuery etc. Our components are supposed to be easy to use even if the end user doesn't have this knowledge. This major drawback is the main motivation for our project, Saab Security haven't found a good alternative to edit their XML-files.

We have found some inspiration regarding presentation of large structures which will be considered when we create our components.

3.4 Investigation of structures and types used by Saab Security

The rawItXml-files, described in section 3.1.1, have a number of types and structures defined that is used in the different XML-files. To present an XML-file in a user friendly and efficient way, these types need to be analyzed. We looked into the rawItXml-files and examined the types closer.

3.4.1 Identified structures and types

The identified types are presented in Table 3.1, along with a description of what each type is.

Type	Description
List	A list is an element with several children of the same type.
Struct	A struct defines a set of children where each child has its own type.
Bool	The bool type is defined in the schema file as having one of the following string values: true, TRUE, false, FALSE, yes, YES, no, NO, 0, 1
String	A simple string.
Regexp	A regular expression used to ensure correct format of a string. The regular expression itself is represented as a string.
Enum	Contains a list of possible values.
int32	A signed 32-bit integer value.
uint32	An unsigned 32-bit integer.
uint16	An unsigned 16-bit integer.
Double	A double number.
Float	A float number.
Choice	One of a set of specified choices. Choices are specified in the <children> node as <elem> nodes.

Table 3.1. *Identified types and structures*

3.4.2 Possible display options

In Table 3.2 a suggested display option is presented for each type. These display options will be used in the main application when writing the presentation components.

Type	Display suggestion
List	Could be displayed as a table. If there are only parameters of one type in the list use single column table. If it is a list of structs use one column for each element in the struct.
Struct	Display as a row in a table. If not part of a list we could use a single row table with one column for each element in the struct.
Bool	Can be displayed as a drop down list with the possible choices.
String	Displayed as a string.
Regexp	Displayed as a string that is evaluated against the regular expression.
Enum	Can be displayed as a drop down list with the possible choices.
int32	Displayed as a signed integer.
uint32	Displayed as an unsigned integer.
uint16	Displayed as an unsigned integer.
Double	Displayed as a double.
Float	Displayed as a float.
Choice	Displayed as the chosen node. I.e. if a struct is chosen, the element is shown as a struct.

Table 3.2. *Suggested display options*

4 Results

This section will cover the results attained during this master thesis. Both results related to the use of Scrum and software components produced will be presented.

4.1 Scrum

Evaluating how much Scrum has helped us during this master thesis is difficult since we don't have any other method to compare with. Also, it is difficult to measure how effective a software development project has been. A measurement like lines of code per hour doesn't say anything about how efficient the development has been.

Thanks to Scrum, we always knew what to do and we got continuous feedback from our supervisor. This meant that we always worked with important and necessary tasks.

4.1.1 Retrospective

After each sprint we evaluated Scrum as development method to improve the process. This evaluation is called sprint retrospective. Due to these retrospective meetings, we were able to improve the process to get the most out of using Scrum.

The document from each retrospective meeting can be found in APPENDIX A.

4.2 Software components

To be able to present the data in customized ways, based on the type of the data, we needed to create a set of components to handle different situations. As requested by Saab we kept a high focus on reusability and tried to separate logic from user interface.

4.3 Presenting semi structured data

XML is a semi structured data format [16]. This means that the data in an XML document do not fit into the relational model. To present this kind of data structure is challenging since it is not possible to decide beforehand how the data can be represented.

In traditional, fully structured, data formats the standard way of presenting data is by using tables. Relational databases for example are designed using a number of tables and specified relations between them. An XML document however cannot always be presented in this way. For instance an XML element can have a number of child nodes that do not have the same structure. An element like that cannot be presented using a table since the child nodes do not fit into the same set of columns. However, it is important to note that in many cases the child nodes do follow the same structure. In other words, XML data can sometimes be presented using tables but you cannot assume that tables can be used.

We have decided to use two different presentation options. We have created two main components to present, navigate and alter the XML structure. For the parts of the structure that can be represented using tables we have created a table component that can be connected to a node and present the contents. In addition to that we have created

a tree view that complements the table structure and gives a good overview of the XML data.

4.3.1 Logic components

These components are pure logic, there is no presentation or GUI parts here. All components described below are located in the namespace XPresXml in our project.

4.3.1.1 Main data structure

.NET has its own XML structures called XmlDocument and XmlNode in System.Xml. For our intents and purposes these structures are too limited in both functionality and customizability. Because of this we chose to handle the XML data in similar data structures that we designed ourselves. We named these structures XPresXmlDocument and XPresXmlNode. When supplied with additional information, from a schema or a rawItXml, a reference to an XPresXmlNodeInfo object, described in detail below, can be added. To increase reusability and to make the use of these new classes more understandable we also created our own exception and event argument classes.

XPresXmlNode

The XPresXmlNode structure is a tree structure where each node keeps track of its own parent and children.

Each node has a number of parameters describing its contents, described in Table 4.1

Children	A list of references to child nodes.
Name	The name of the node.
Node info	A reference to an XPresXmlNodeInfo object.
Node type	The type of the contents in the node.
Parent	A reference to the parent.
TreeText	A descriptive text to make identification of the node easier for the user.
Value	The value of the node.
XmlNodeType	The type of the node.

Table 4.1. *Parameters of XPresXmlNodes*

XPresXmlNodeInfo

The XPresXmlNodeInfo structure stores information about each node. It specifies what name the node must have, what type of values it should contain and several other constraints described below. This information is extracted from either an XML schema or from a rawItXml[1] file. Regardless of the fact that these files don't resemble each other they contain the same information, except for information regarding attributes.

The information stored by each XPresXmlNodeInfo element is shown in Table 4.2

[1] See chapter 3.1.1

Children	A list of references to XPresXmlInfo-objects with information about what children the node can have.
Comment	Comment with extra explanation about the node.
Enum_values	If the value of the node is limited to only a few possible values, this parameter stores a list of these values.
It_type_name	Describes the type of data structure, if any, the node represents. E.g. list(string), set(int32) etc.
Max_occurs	Maximum number of occurrences of the node.
Min_occurs	Minimum number of occurrences of the node.
Max_value	The maximum value the node can have.
Min_value	The minimum value the node can have.
Name	The name of the node.
Node_type	The type of the contents in the node. Possible types described in Table 3.2.
Parent	A reference to the XPresXmlNodeInfo object with information about the node's parent.
Regular_Expression	If the regular expression parameter is set the node must match the specified pattern [19].
Schema_type_name	If the node type is type_reference, this parameter stores the name of the referred XPresXmlNodeInfo-object.

Table 4.2. *Parameters of XPresXmlNodeInfo*

Relationship between XPresXmlDocument and XPresXmlNodeInfo

If a schema has been supplied, each node needs the ability to access the information that concerns that node. This could be solved by simply saving this information in the node. That solution has some major disadvantages though. There would be a lot of redundant information stored as all nodes with the same "schema template" would keep its own copy of the schema information.

We solved this by instead adding references to the XPresXmlNodeInfo structure. I.e. every node has a reference to a node in the XPresXmlNodeInfo structure. In this way all nodes with the same "schema template" points to the same XPresXmlNodeInfo node as illustrated in Figure 4.1

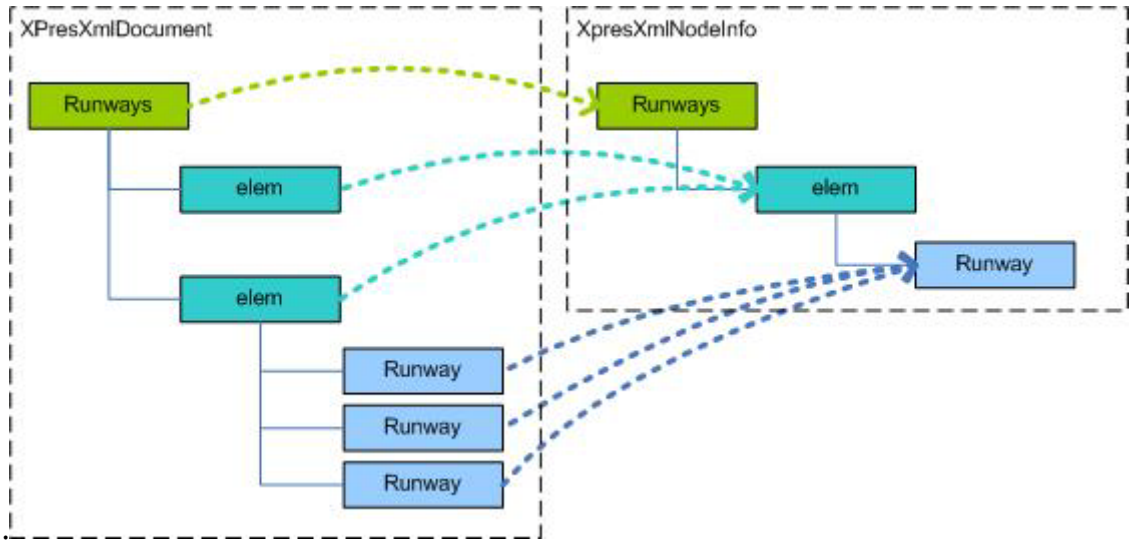

Figure 4.1. *The relationship between the XPresXmlNodes and the XPresXmlNodeInfo nodes*

4.3.1.2 WordPrinter

This class provides methods that allow the user to create a Microsoft Word document containing a table, from a .NET System.Data.DataTable. This feature was specifically requested by employees at Saab Security to simplify the creation of customer friendly presentation of configuration data. Included in this class is functionality to specify which columns to include, when creating the table. This allows the user to exclude unwanted or unimportant information.

4.3.1.3 CsvPrinter

The CsvPrinter class is very similar to the WordPrinter but instead of generating a Word document, a comma separated values-file [17], or .csv-file, is created. The character encoding used is the system default on the user's system. This is a simple format that can easily be opened in Microsoft Excel, and other spreadsheet applications. A developer using this class can specify a certain string to use as delimiter in the output file. Like WordPrinter, this class was requested by employees at Saab Security to simplify data presentation.

4.3.1.4 LogicExpressionParser

In a couple of Saab Security's configuration files there is a special structure to describe boolean logic expressions. This structure is very difficult to edit and gets very complex as the expressions grow. An example expression in its XML format is shown in Example 4.1.

```
<and_condition>
  <X>1</X>
  <or_condition>
    <Y>2</Y>
    <not_condition>
      <Z>3</Z>
    </not_condition>
  </or_condition>
  <W>4</W>
</and_condition>
```

Example 4.1. *Example of boolean expression in XML document*

We wanted to present this quite unreadable structure like an algebraic expression. E.g. the expression in Example 4.1 should be shown as (X=1 **AND** (Y=2 **OR** (**NOT** Z=3)) **AND** W=4). We also felt that we wanted the ability to easily convert between these two representations.

The LogicExpressionParser class contains a set of methods to convert between the formats described above. To allow reuse together with other components in this project we also created methods to convert the expressions to and from the XPresXmlNode structure.

The result is a set of tools that allow the user to select an XPresXmlNode and convert it to a more readable string format. The user can then alter the string and convert back to the XPresXmlNode structure and from there get the pure XML text representation again.

4.3.1.5 NodeSearch

To allow the user to find nodes fast and easy we created a search method. This method takes a node and a string and searches through the specified node's children to find any matching nodes. The result from this method is a list of nodes matching the search. It is also possible to specify what to search for, i.e. elements, comments and attributes. It's also possible to set whether to search for name of the node or the value of the node or even both.

4.3.1.6 XPresInfoTreeClient

Some of Saab Security's systems run a service called "InfoTree" that allows remote connection and the ability to change the settings of a specific system during runtime. This service provides an interface that you can connect to programmatically from C++ and C# programs. After connecting to the server the developer can get the settings in the form of a tree structure. The XPresInfoTreeClient component provides methods to connect to such a service, through the ComCore[2] protocol. After connecting, the tree structured retrieved from the server is converted to the XPresXmlNode structure that can utilize the other components developed in this project. The component also provides methods to post changes back to the server.

[2] See chapter 3.1.2

4.3.2 User interface components

We have used Windows Forms [18] as our GUI API. There are essentially two different component types in the Windows Forms API: forms and controls.

A form is basically a window. When using Windows Forms a program with a graphical user interface is always encapsulated inside a form. In its basic form it is an empty window in which the developer can add controls.

A control is an element that can be placed in other controls or forms. A control can be anything from a simple text label to a container with hundreds of other controls.

We have developed a number of controls and forms in this project. They have been developed in a way that allows for high reusability with a lot of focus on the public interfaces used to configure them for different situations and to insert and extract data from them. We have also tried to minimize the amount of intelligence in the GUI components, when complex operations were needed we tried to move that code to separate classes.

4.3.2.1 Dialogs

We chose to create a number of dialogs to handle certain actions. For instance, they can be shown when the user clicks on a menu item.

ConnectInfoTree

The ConnectInfoTree dialog provides a user interface for the XPresInfoTreeClient component. It allows the user to specify a ComCore[3] service to connect to and after connecting the user is presented with an XPresTree to navigate the settings structure and an XPresGridView to change settings. When the user wants to commit changes this is done by clicking a button.

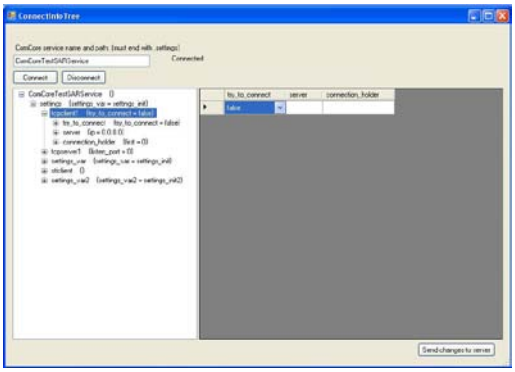

Figure 4.2. *The ConnectInfoTree dialog*

WordWriterDialog

The WordWriterDialog takes a DataTable when created. It presents the user with one check box per column, allowing the user to choose which columns to include. It has a field for the output filename, which opens an open file dialog when it is clicked. When the user has selected columns and specified a filename he or she can click the "Create document" button. The WordWriterDialog will then use the methods in the WordPrinter class to create a Microsoft Word document based on the user's specifications.

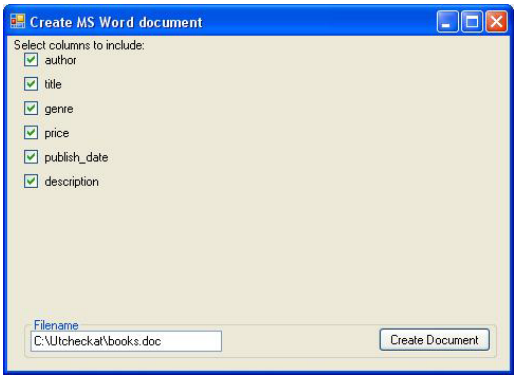

Figure 4.3. *The WordWriter dialog*

[3] See chapter 3.1.2

When the document has been created a dialog indicating this will be shown.

CsvWriterDialog

The CsvWriterDialog is almost identical to the WordWriterDialog since they have almost the same purpose. Of course it uses the methods from CsvPrinter instead of WordPrinter. It saves to a file with the file extension .csv unless the user specifies something else.

InputDialog

The InputDialog is a simple dialog used for adding attributes and comments to the structure. As parameters the InputDialog takes the parent node and the type of node to be created. When clicking on the create button the node is created and added as a child of the node in the input.

EditDialog

EditDialog looks exactly the same as InputDialog, but instead of creating new nodes it is used for editing values of existing nodes.

NewXPresNodeDialog

NewXPresNodeDialog works in the same manner as InputDialog but is for creating elements. When creating a new element the dialog takes advantage of the XPresXmlNodeInfo and adds an input box for all possible children to the new element. This way, several nodes are created when clicking on Create Element button.

If the user wants he could deselect the nodes he doesn't want to create, unless the constraints says the node must exists, i.e. has a min occurs greater than zero. Value and type constraints for each new node are also respected. This means it's impossible to create an element that doesn't follow the rules specified in the schema or rawItXml file.

SearchDialog

The SearchDialog gives the user a graphical interface to search for nodes. In this graphical interface the user can specify what to search for and how to search for it. SearchDialog uses the NodeSearch method, with parameters from the graphical interface, and displays the results in a list. From the list of results, the user can select, edit or delete the node.

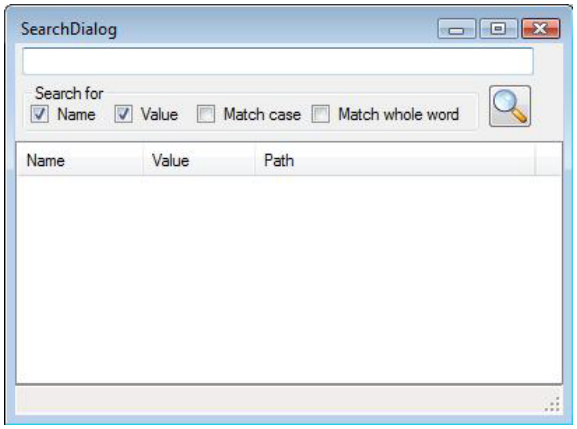

Figure 4.4. *SearchDialog*

InvalidNodesDialog

InvalidNodesDialog is a dialog that searches through the document for nodes that doesn't match the schema or rawItXml-file provided. The nodes will be listed together with an explanatory text describing why the node doesn't match the schema or rawItXml-file. Similarly to SearchDialog, you can also select, edit and delete nodes.

4.3.2.2 Controls

All controls works independently of other controls. It's up to the program designer to add the controls he would like to use.

Some of the controls implements an interface called IConnectable, this interface specify methods that must exists so that the components can be connected to each other. When connected to each other, all controls listen on certain events from the other controls and updates when the structure has been changed. This means adding controls to a new program and connecting them takes very little effort.

LogicExpression

The LogicExpression component, shown in Figure 4.5, provides a user interface for the LogicExpressionParser. It is created by supplying it with a node containing an "and_expression" or an "or_expression". The control presents the user with two multiline textboxes. The topmost box is activated for editing and it is in this box the user should edit the expression that he/she wants to convert to a boolean expression in XML, in the format used by Saab Security.

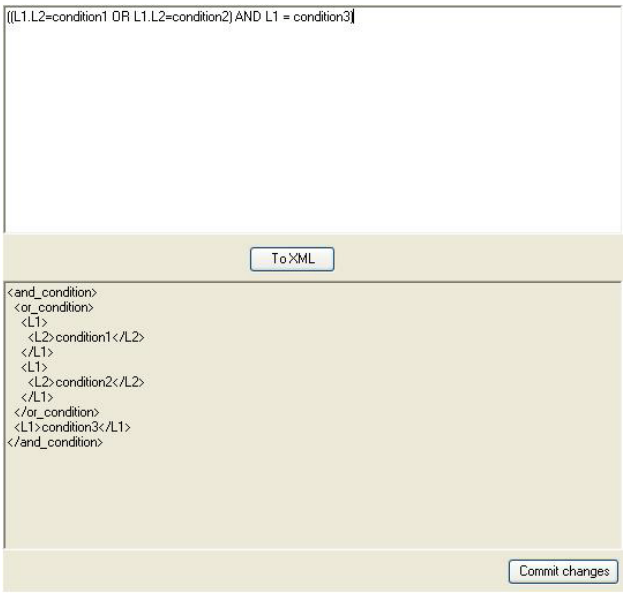

Figure 4.5. *The LogicExpression control*

The expression must be well formed with regard to parentheses. The entire expression must be encapsulated in parentheses and each sub expression must also be encapsulated in parentheses. If a user enters an expression that does not follow these rules the text turns red and conversion to XML text is not possible.

In the bottom textbox the user is presented with the XML text corresponding to the expression in the upper textbox. As the user edits the expression the bottom textbox updates the XML representation in real time.

When the user is satisfied with the expression he/she can click "Commit changes" button and the node used to create the control is updated to the new expression.

XPresGridView

XPresGridView, shown in Figure 4.6, displays the XML structure in table form. Showing a list in table form gives the user a great overlook of the values stored in the list. The table is not

	author	title	genre	price	publish_date	description
▸	Gambardella, Matthew	XML Developer's Guide	Computer	44.95	2000-10-01	An in-depth look at creat
	Ralls, Kim	Midnight Rain	Fantasy	5.95	2000-12-16	A former architect battles
	Corets, Eva	Maeve Ascendant	Fantasy	5.95	2000-11-17	After the collapse of a na
	Corets, Eva	Oberon's Legacy	Fantasy	5.95	2001-03-10	In post-apocalypse Engla
	Corets, Eva	The Sundered Grail	Fantasy	5.95	2001-09-10	The two daughters of Ma
	Randall, Cynthia	Lover Birds	Romance	4.95	2000-09-02	When Carla meets Paul a
	Thurman, Paula	Splish Splash	Romance	4.95	2000-11-02	A deep sea diver finds tr
	Knorr, Stefan	Creepy Crawlies	Horror	4.95	2000-12-06	An anthology of horror st
	Kress, Peter	Paradox Lost	Science Fiction	6.95	2000-11-02	After an inadvertant trip t
	O'Brien, Tim	Microsoft .NET: The Programming Bible	Computer	36.95	2000-12-09	Microsoft's .NET initiative
	O'Brien, Tim	MSXML3: A Comprehensive Guide	Computer	36.95	2000-12-01	The Microsoft MSXML3 p
	Galos, Mike	Visual Studio 7: A Comprehensive Guide	Computer	49.95	2001-04-16	Microsoft Visual Studio 7
*						

Figure 4.6. *The XPresGridView control*

only for showing nodes, editing values in a cell is also possible. Each non-empty cell and row contains a pointer to an XPresXmlNode. This means that editing, or creating new rows, automatically updates the values stored in the structure.

Nodes that contain enumerators are shown as a drop down list with the possible options. Nodes representing numbers, lists or have a regular expression set are also represented by special cells that we have designed to make sure the user doesn't input invalid data.

The XPresGridView works regardless if the node has an XPresXmlNodeInfo set or not. However without the extra information about the node, several functions, such as specialized cells, are not enabled. The XPresGridView can show all nodes in a well formed XML documents, but some nodes are better represented by another control. How and when the XPresGridView is used is up to the implementer of the control.

XPresTree

XPresTree, shown in Figure 4.7, displays the XML structure in tree form. This gives a great overview of the complete structure of the XML file. With this control, the user has the possibility to select, edit, delete and create new nodes.

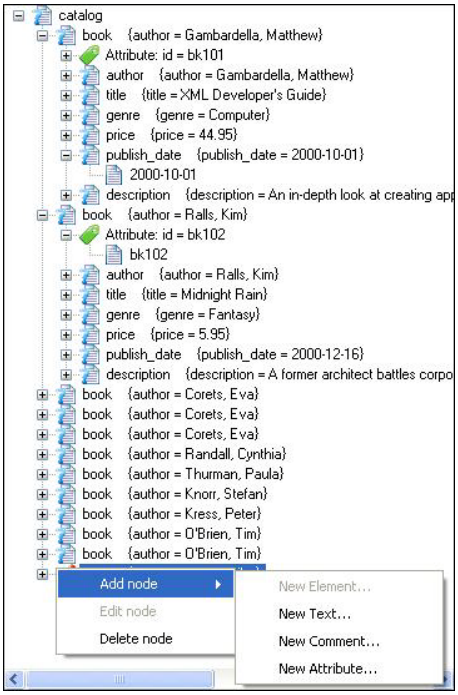

Figure 4.7. *The XPresTree control*

Creating nodes are done by right clicking a node and select Add Node from the menu shown. Adding nodes uses the NewXPresNodeDialog and InputDialog to create the new node. The XPresTree has the same functionality whether a schema, or rawItXml file, is loaded or not.

NavigationToolStrip

The NavigationToolStrip, shown in Figure 4.8, is a tool strip with buttons for navigating through the structure. It consists of buttons for navigating back, forward and up and a text box displaying current path. It also contains buttons for undoing and redoing changes and for searching for a node.

Figure 4.8. *The XPresNavigationToolStrip control*

The amount of history stored in the control is defined by the program designer when adding the control to his program.

4.3.3 Viewing the components in a web browser

To give ideas on how our components could be used we created a web page with our controls. This was done by creating a Windows Presentation Foundation [20], WPF, browser application that could host our components. The WPF application use a

language called Extensible Application Markup Language [21], XAML, which is a language based on XML [1] created to define user interface elements. Many controls available for Windows Forms are also available in WPF. WPF also contains a control, called WindowsFormsHost [22], which can be used to host any windows forms control, including those we have created. This gave us the ability to insert our components, without modification and with full functionality, into a web page.

We see the web page as a great choice for customers that don't want to, or aren't allowed to, install or run new applications on their computers. Besides a browser and an internet connection, the user still needs to have .NET installed to be able to view web pages that are hosting WPF browser applications.

4.3.4 Testing

Saab Security sometimes requires certain test documents to be created along with new products and components. We chose to create these documents for the most important components to ensure quality, to identify bugs and to maintain functionality when adding features.

The documents, called Software Test Descriptions or STDs, describe tests to be performed on the component as a part of the software testing phase of the development. Each component has its own document that describes how to perform tests on that component as well as what is needed to perform the tests.

To complement the STD documents we created another type of document. This document, called Software Test Report (STR), is a checklist that can be used when going through the tests described in one or several STDs. Each test case is marked as successful or failed, along with comments if appropriate.

5 Discussion

Below we will discuss the results achieved in the project. We will discuss whether we have reached our goals concerning usability and reusability. We will also discuss how it has been to work with Scrum in an academic project.

5.1 Software components

The goal of this project was to create a set of components that would simplify editing of XML configuration files in a way that allows for reuse of the components. To ensure that our components will be used after the project was finished we have done our best to make sure that they are reusable in as many situations as possible as well as easy to use.

5.1.1 Reusability

Throughout the project we have created a lot of components that were specialized for a certain context. When realizing that the reusability factor has decreased significantly we tried to separate the parts that could be reused from the parts that were context specific. In this way we ensure that the resulting components will be of much use to Saab Security.

One method we used, to increase reusability, was to provide a lot of alternative ways to use the components. We provide different class constructors allowing the developer to use our components in different situations with different types of available data. Similarly we have made sure to provide methods that can be used with different input parameters depending on data available at the call site.

Another separation we made was to separate data processing and algorithms from the presentation logic. The structure we aimed for was to write the presentation logic components in a way that requires as little computation as possible. Instead when computation was needed there should be a component in the non-presentation library that can perform it. This separation allows for reuse of computationally advanced components in projects and situations where presentation logic is not needed or desired.

5.1.2 Usability

We wanted our components to be easy to use for an end user as well as for a developer. To increase usability for the end user we have made sure to include tooltips and descriptive texts for all parts that might be confusing. We have also made sure to follow de facto standards such as certain shortcut combinations and menu setups, this should make the user feel more familiar with the components and quickly understand how to use them.

Increasing the usability for the developer is a quite different challenge. One important aspect is the documentation. We have used C#'s built in XML documentation notation [23] to comment the classes and methods written. This provides the developer with descriptions of different parts directly from Visual Studio. We have also created a detailed documentation section on Saab Security's internal wiki that can be used for in depth reference of the components as well as simple "getting started" tutorials.

As a step to ensure we had achieved good usability we let employees at Saab Security try to use our components both as developers and as end users. This produced extremely valuable feedback which we could then use in the further development.

After we adjusted the components based on that feedback the employees as Saab Security were very pleased with the work we had performed. We got a lot of positive remarks concerning the speed at which new applications could be developed using our components. There were also positive remarks towards the usability for end users, in particular the fact that the end users do not need any knowledge about XML to successfully and securely edit advanced XML configuration files.

5.2 Development method

The second of our goals was to evaluate the use of Scrum in a semi academic project. Throughout the project we have made sure to follow the principles we decided on from the beginning.

Sometimes a certain step in the process felt a bit unnecessary. For instance a meeting attended by only the two of us, who have been sitting 2 meters away from each other for the entire project, felt almost ridiculous. However, after a while we realized the importance of these meetings. The meetings gave us time to make sure that we both were on the page about the different parts of the project. They also resulted in documents that we could refer to if any uncertainties arose.

One of the fundamental parts of Scrum is the aim to have a deliverable product by the end of each sprint. In our case the deliverable product would not only be the software components, but also the project report and other documentation. By aiming for a deliverable product in each sprint we were motivated to work continuously on all parts of the project, not only the development parts. By the end of sprint one, we already had a report of 26 pages and a couple of working software components.

Another part of Scrum that had a significant impact on the success of this project was the planning structure. Each 3 week sprint was planned in detail with clearly stated sub goals. Each of these goals, or backlog items, was distinctively separated from the other. This gave us an incredible ability to work in parallel. We could work on a separate item each and not risk doing the same work twice, or make each other's goals impossible. At the end of each sprint we held a demo to which all the colleagues of the Saab Security ATM department in Gothenburg were invited. These demo sessions had good attendance and we received a lot of constructive criticism and suggestions for upcoming sprints. This way of structuring our work has made our production rate very high and made sure we stayed on track towards a product that would be useful to our colleagues.

The separation of the different roles in Scrum is another aspect that fits well into a semi academic project. In the beginning of the project we decided to let our supervisor at Saab, Erik Jonsson, assume the role of product owner. This meant that he is only involved at the beginning and end of each sprint. This made us self sustaining for the duration of a sprint, increasing our productivity while decreasing the work load for our supervisor.

6 Conclusions

In this master thesis we have created a set of components, which would simplify XML editing, with high usability and reusability. We have also evaluated Scrum as a part of an academic project.

6.1 Software components

The components we have developed form a competent toolset for editing and handling XML-files and structures. We have achieved a high degree of reusability that will make the components more likely to be used in different situations.

To illustrate the capability of the components we created a demo project that displays most of the available features. When demonstrating this demo project to colleagues at Saab Security we have received a lot of positive feedback. We have definitely achieved the goal we set up at the beginning of the project and Saab Security is positive regarding the resulting product.

We also let a college test how it is to work with our components, to see how easy it is to create solutions based on our components. We were overwhelmed with how quickly she understood our components and were able to create a working solution. This proves that we achieved both high reusability and high usability.

6.2 Development method

The use of the Scrum development method during our project proved a great success. Our colleagues remarked at several occasions that they were impressed with the high progress rate we achieved. Sometimes we were even surprised ourselves at how much work we got done during a three week sprint.

Scrum is well suited for projects performed in small groups as well as projects with academic parts included. We truly believe that most thesis, and similar, project groups would benefit a lot from using this method. The strong result focus as well as the task breakdown makes the work not only very efficient, but even more important it makes it more fun to do the work.

7 References

[1] World Wide Web Consortium, “Extensible Markup Language (XML) 1.0 (Fifth Edition),” *W3C*, Nov. 26, 2008. [Online]. Available: http://www.w3.org/TR/REC-xml/. [Accessed: Feb. 22, 2010]

[2] Cover Pages, “XML Applications and Initiatives,” *Cover Pages*, June 25, 2005. [Online]. Available: http://xml.coverpages.org/xmlApplications.html. [Accessed: Feb. 22, 2010].

[3] RSS Advisory Board, “RSS 2.0 Specification,” Mar. 30, 2009. [Online]. Available: http://www.rssboard.org/rss-specification. [Accessed: Feb. 22, 2010].

[4] World Wide Web Consortium, “XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition),” *W3C*, Aug. 1, 2002. [Online]. Available: http://www.w3.org/TR/xhtml1/. [Accessed: Feb. 22, 2010].

[5] Rob Weir, "OpenDocument Format: The Standard for Office Documents," *IEEE Internet Computing*, vol. 13, no. 2, pp. 83-87, Mar./Apr. 2009

[6] K. Schwaber, *Agile Project Management with Scrum*. Microsoft Press, 2004.

[7] Microsoft Corporation, “MFC Reference,” *MSDN Visual C++ Developer Center*, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/d06h2x6e.aspx. [Accessed: Feb. 22, 2010].

[8] Microsoft Corporation, “COM Interop,” *MSDN Visual C++ Developer Center*, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/6bw51z5z.aspx. [Accessed: Feb. 22, 2010].

[9] Microsoft Corporation, “Hosting a Windows Form User Control as an MFC Dialog Box,” *MSDN Visual C++ Developer Center*, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/676cbawx.aspx. [Accessed: Feb. 22, 2010].

[10] Liquid Technologies, “Liquid XML Studio 2010 Overview,” *Liquid XML Studio*, 2010. [Online]. Available: http://www.liquid-technologies.com/Xml-Studio.aspx. [Accessed: Feb. 22, 2010].

[11] SyncRO Soft Ltd, “<oXygen/> XML Editor version 11.1,” *XML Editor*, 2010. [Online]. Available: http://www.oxygenxml.com/. [Accessed: Feb. 22, 2010].

[12] Oracle Corporation, “NetBeans,” *NetBeans IDE 6.8*, 2010. [Online]. Available: http://netbeans.org/. [Accessed: May 17, 2010]

[13] Altova, “XML Editor, Data Management, UML and Web Services tools,” *XML Editor, Data Management, UML and Web Services tools from Altova*, 2009. [Online]. Available: http://www.altova.com/. [Accessed: Feb. 22, 2010].

[14] Syntext, Inc., “Serna Free - Open Source XML Editor,” *Serna Free - Open Source XML Editor*, 2009. [Online]. Available: http://www.syntext.com/products/serna-free/. [Accessed: Feb. 22, 2010].

[15] XPontus Team, “XPontus,” *XPontus - Homepage*, 2008. [Online]. Available: http://xpontus.sourceforge.net. [Accessed: Feb. 22, 2010].

[16] C. M. Sperberg-Mcqueen, ”XML and semi structured data,” *World Wide Web Consortium*, Oct 2005.

[17] Y. Shafranovich, "Common Format and MIME Type for Comma-Separated Values (CSV) Files," *The Internet Engineering Task Force (IETF)*, Oct. 2005. [Online]. Available: http://tools.ietf.org/html/rfc4180. [Accessed: Apr. 15, 2010].

[18] Microsoft Corporation, "Windows Forms," *MSDN Visual Studio Developer Center*, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd30h2yb.aspx. [Accessed: Apr. 15, 2010].

[19] J. Friedl, *Mastering Regular Expressions*. O'Reilly Media, 2006

[20] Microsoft Corporation, "Introduction to WPF," *MSDN Visual Studio Developer Center*, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa970268.aspx. [Accessed: Apr. 26, 2010].

[21] Microsoft Corporation, "XAML Overview (WPF)," *MSDN Visual Studio Developer Center*, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms752059.aspx. [Accessed: Apr. 26, 2010].

[22] Microsoft Corporation, "Windows Forms – WPF Interoperability FAQ," *WindowsClient.NET*, 2010. [Online]. Available: http://windowsclient.net/learn/integration.aspx. [Accessed: Apr. 26, 2010].

[23] Microsoft Corporation, “XML Documentation Comments (C# Programming Guide),“ *MSDN Visual Studio Developer Center*, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/b2s063f7.aspx. [Accessed: Apr. 28, 2010].

APPENDIX A - Sprint retrospective

These documents are the results of each sprint retrospective meeting.

A.1 Sprint 1

The results from the retrospective after sprint 1 are presented below.

A.1.1 Positive

- Surprisingly good progress on the final report.
- Good size of the backlog items. Small enough to give a good overview of what should be done yet not so small that they felt unnecessary.

A.1.2 Negative

- Pessimistic time estimations created a shortage of work towards the end of the sprint.
- Some backlog items were too vaguely defined which resulted in difficulty to understand what that item really meant.

A.1.3 Changes for the next sprint

- Adjust time estimations towards more optimistic times.
- Define backlog items more explicitly and make sure everyone involved agree on what each item means.

A.2 Sprint 2

The results from the retrospective after sprint 2 are presented below.

A.2.1 Positive

- Lots of positive reactions from colleagues after the sprint demo.
- The change concerning more explicitly defined backlog items gave good results.
- The separation of the backlog items is working well, making it easy to work on one item each without unnecessary waiting for another part to be finished.

A.2.2 Negative

- We over estimated our productivity during planning, resulting in a few points left undone after the sprint.
- Drifted away from the tasks of certain backlog items. For instance we spent time on fixing new bugs and loosing focus on the backlog item at hand.
- The possibility to perform certain backlog items was impossible because of factors that were out of our control. E.g. colleagues unavailable etc.

A.2.3 Changes for the next sprint

- We choose to not adjust time estimations but instead try to raise the focus on the backlog item we are currently working on.
- All in all we are satisfied with the previous sprint and do not want to change too much.

A.3 Sprint 3

The results from the retrospective after sprint 3 are presented below.

A.3.1 Positive

- Time estimations where almost correct. The decision not to adjust time estimation but instead raise the focus on the current item proved to be a good decision.

- The end product starts to shape, we took several steps forward during the sprint.

A.3.2 Negative

- One of the backlog items was removed due to circumstances we couldn't foresee. The colleague who proposed the item realized that the item was not needed due to changes in other projects.

- Slow progress with the report.

A.3.3 Changes for the next sprint

- Put a bigger focus on the report since progress was really slow this sprint. The report should be included in the idea with having a deliverable product after each sprint.

A.4 Sprint 4

The results from the retrospective after sprint 4 are presented below.

A.4.1 Positive

- Great progress with the report; we finally achieved the goal of having a report that theoretically could be handed in.
- The end product starts to shape, we took several steps forward during the sprint.

A.4.2 Negative

- Time estimations where too optimistic, probably because of loss of focus of the current item. One example is that we spent a lot of time fixing bugs when we only should have tested the components.
- We didn't do all backlog items, some where because of our time estimations while others where because of colleges being unavailable.

A.4.3 Changes for the next sprint

- Clearly separate bug-fixing from actual development.

A.5 Sprint 5

The results from the retrospective after sprint 5 are presented below.

A.5.1 Positive

- We completed several minor items that we wanted to do for a long time.
- We accomplished a great deal despite the mediocre planning.
- The product is ready to be delivered to Saab Security.

A.5.2 Negative

- While planning we realized that we couldn't plan for the whole sprint. Mainly because we needed input from a college.

A.5.3 Changes for the next sprint

Since this was the last sprint, there is no need for changes.